Universidade Federal de Alagoas (UFAL)
Campus Arapiraca


# Programação Orientada a Objetos (POO)

## 06 - Herança e polimorfismo

**Alexandre de Andrade Barbosa**

alexandre.barbosa@arapiraca.ufal.br

# Objetivos

## Objetivos

- Apresentar os conceitos de herança e polimorfismo
- Apresentar o conceito de classes abstratas
- Descrever modificador abstract quando aplicado a classes e métodos
- Apresentar o conceito de herança simples e herança múltipla
- Descrever o que é uma Interface em Java
- Exibir exemplos ilustrativos para tais conceitos

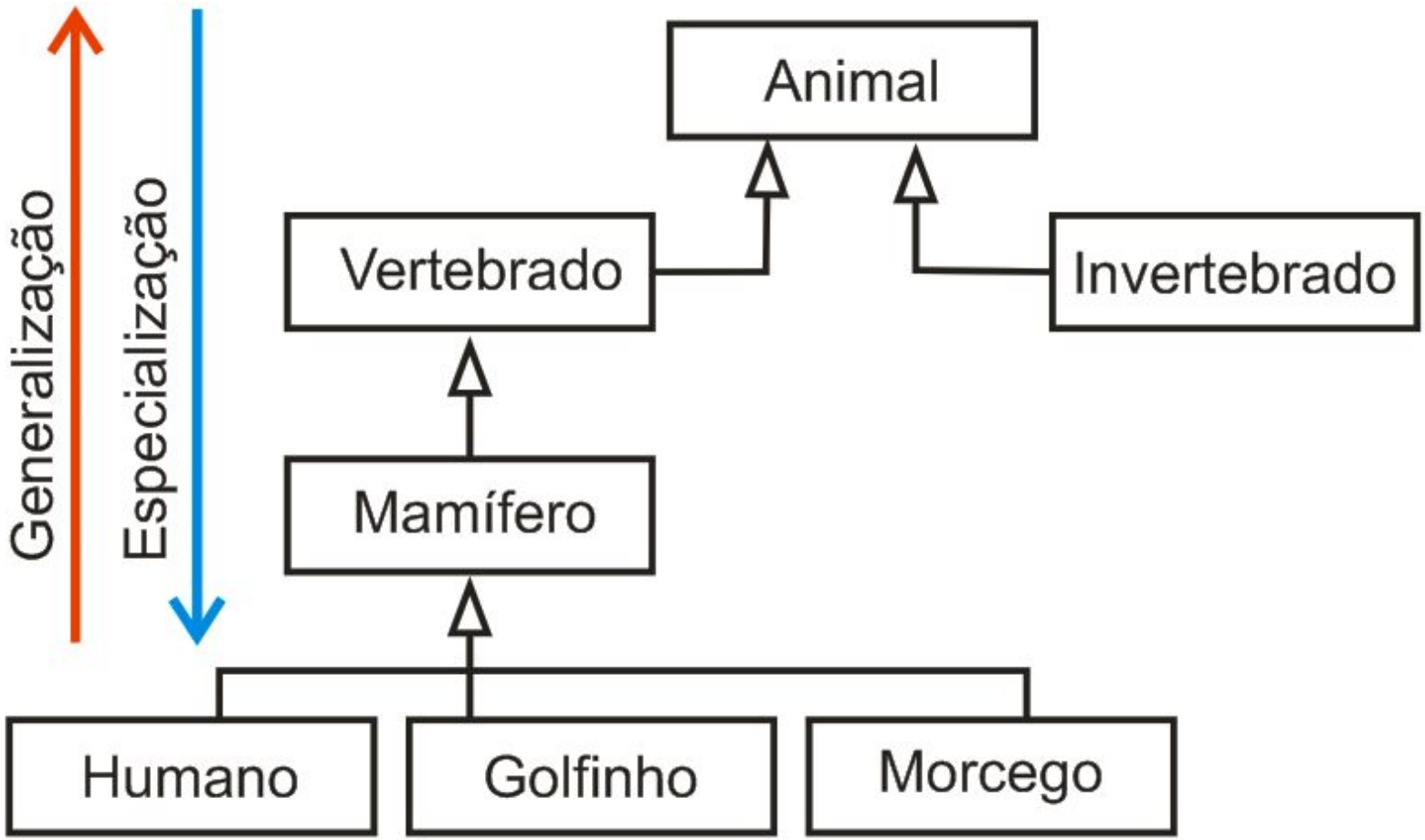
# Herança e polimorfismo

## Herança

- Herança é um dos mecanismos existentes em O.O. para se obter reusabilidade.
- É possível criar uma classe a partir de outra já existente.
- Define-se características e ações comuns em classes mais gerais, e adiciona-se características e ações em classes mais específicas.

# Herança e polimorfismo

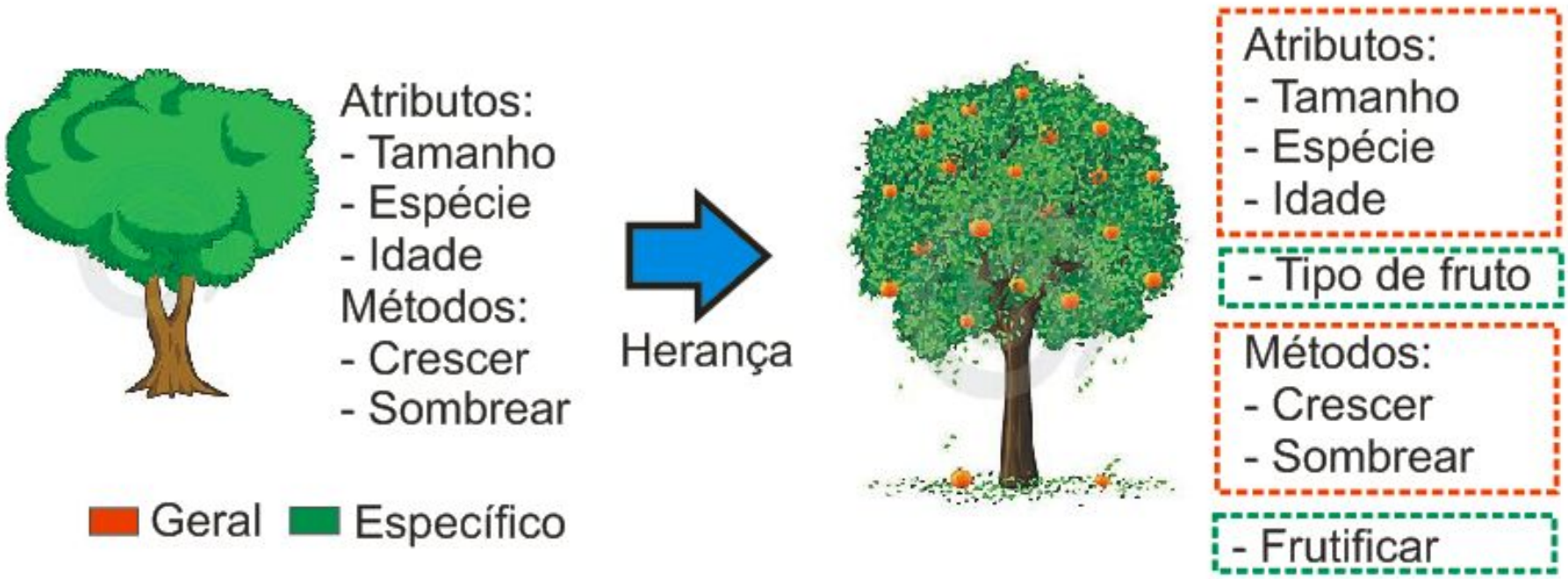
## Herança



# Herança e polimorfismo

## Herança

Uma analogia...



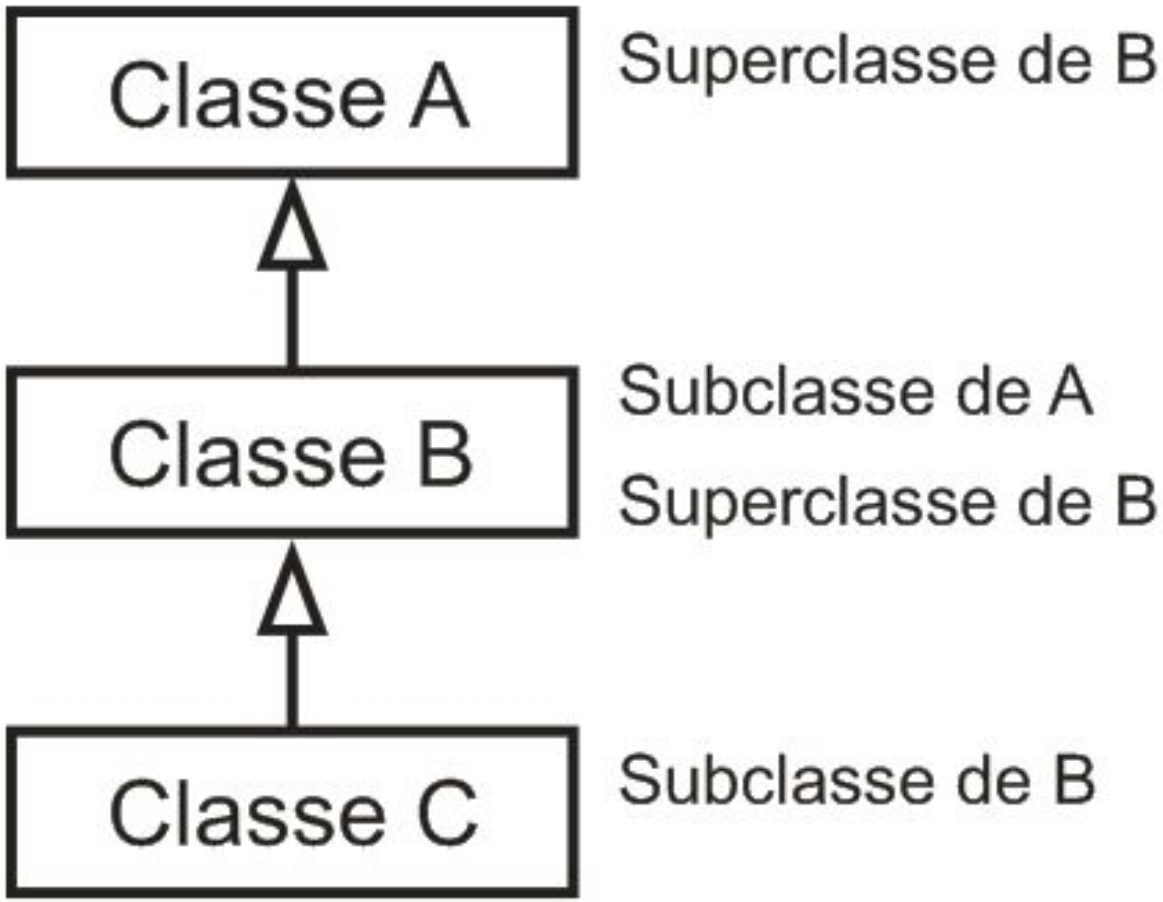
# Herança e polimorfismo

## Herança

- A classe mais geral é denominada super-classe, classe-pai ou classe-mãe.
- A classe mais especifica é denominada sub-classe ou classe-filha.

# Herança e polimorfismo

## Herança



# Herança e polimorfismo

## Herança

```
1  public class Pessoa {
2      private String nome;
3      public Pessoa(String nome) {
4          this.nome = nome;
5      }
6      public String getNome() {
7          return nome;
8      }
9      public void setNome(String nome) {
10         this.nome = nome;
11     }
12 }
```

# Herança e polimorfismo

## Herança

```
1 // extends permite a herança
2 public class Aluno extends Pessoa {
3     public Aluno(String nome) {
4         // referência ao construtor da superclasse
5         super(nome);
6     }
7 }
```

```
1 // extends permite a herança
2 public class Professor extends Pessoa {
3     public Professor(String nome) {
4         // referência ao construtor da superclasse
5         super(nome);
6     }
7 }
```

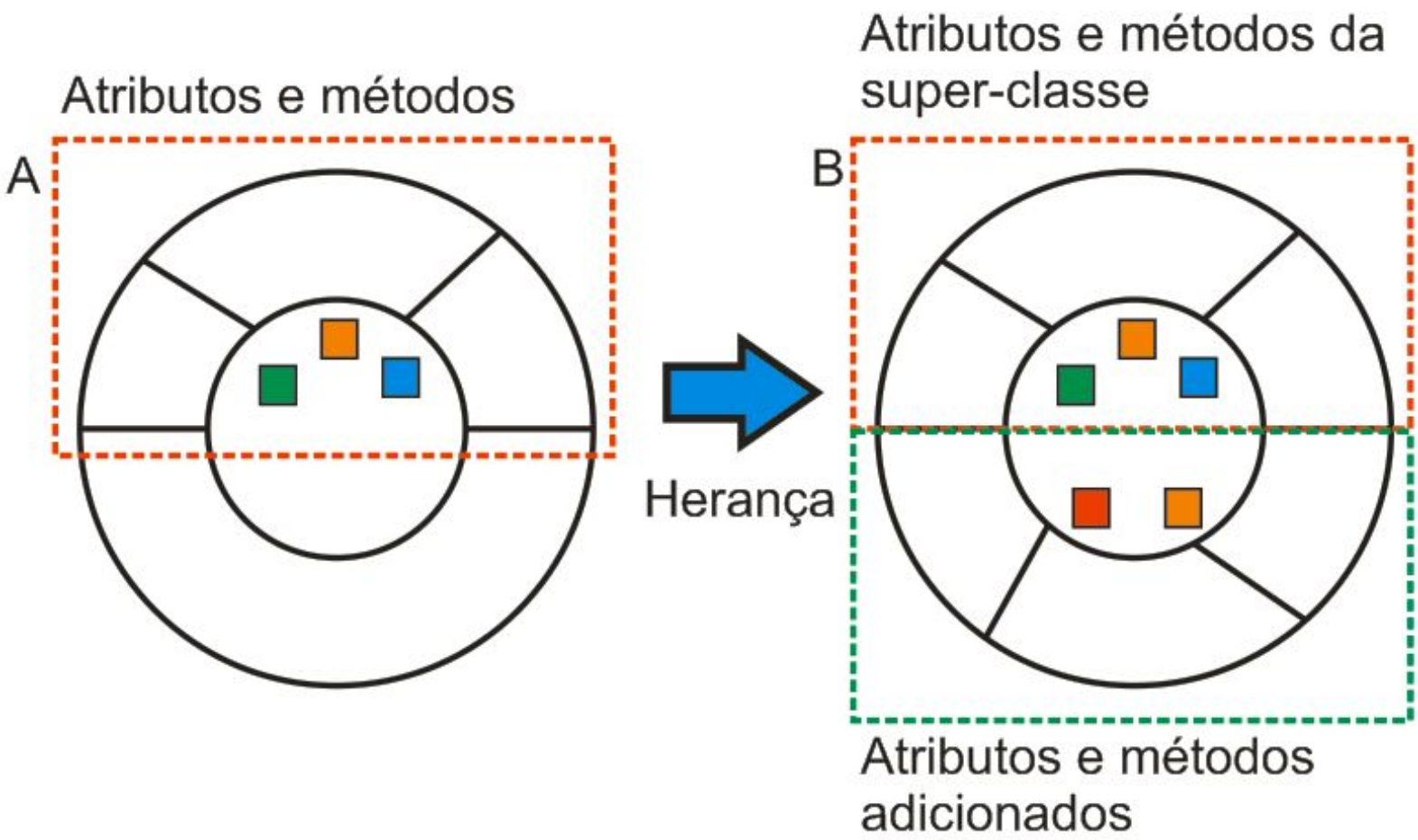
# Herança e polimorfismo

## Herança

```
1  public class Main {
2      public static void main(String[] args) {
3          // Instanciando um objeto do tipo Pessoa
4          Pessoa p = new Pessoa("João");
5          System.out.println(p.getNome());
6
7          // Instanciando um objeto do tipo Aluno
8          // Aluno é subclasse de Pessoa
9          Aluno a = new Aluno("Maria");
10         // chamada a um método definido na superclasse
11         System.out.println(a.getNome());
12
13         // Instanciando um objeto do tipo Professor
14         // Professor é subclasse de Pessoa
15         Professor prof = new Professor("José");
16         // chamada a um método definido na superclasse
17         System.out.println(prof.getNome());
18     }
19 }
```

# Herança e polimorfismo

## Herança



# Herança e polimorfismo

## Herança

```
1  public class Aluno extends Pessoa {
2      private String matricula;
3
4      public Aluno(String nome, String matricula) {
5          super(nome);
6          this.matricula = matricula;
7      }
8
9      public String getMatricula() {
10         return matricula;
11     }
12
13     public void setMatricula(String matricula) {
14         this.matricula = matricula;
15     }
16 }
```

# Herança e polimorfismo

## Herança

```
1  public class Professor extends Pessoa {
2      private float salario;
3
4      public Professor(String nome, float salario) {
5          super(nome);
6          this.salario = salario;
7      }
8
9      public float getSalario() {
10         return salario;
11     }
12
13     public void setSalario(float salario) {
14         this.salario = salario;
15     }
16 }
```

# Herança e polimorfismo

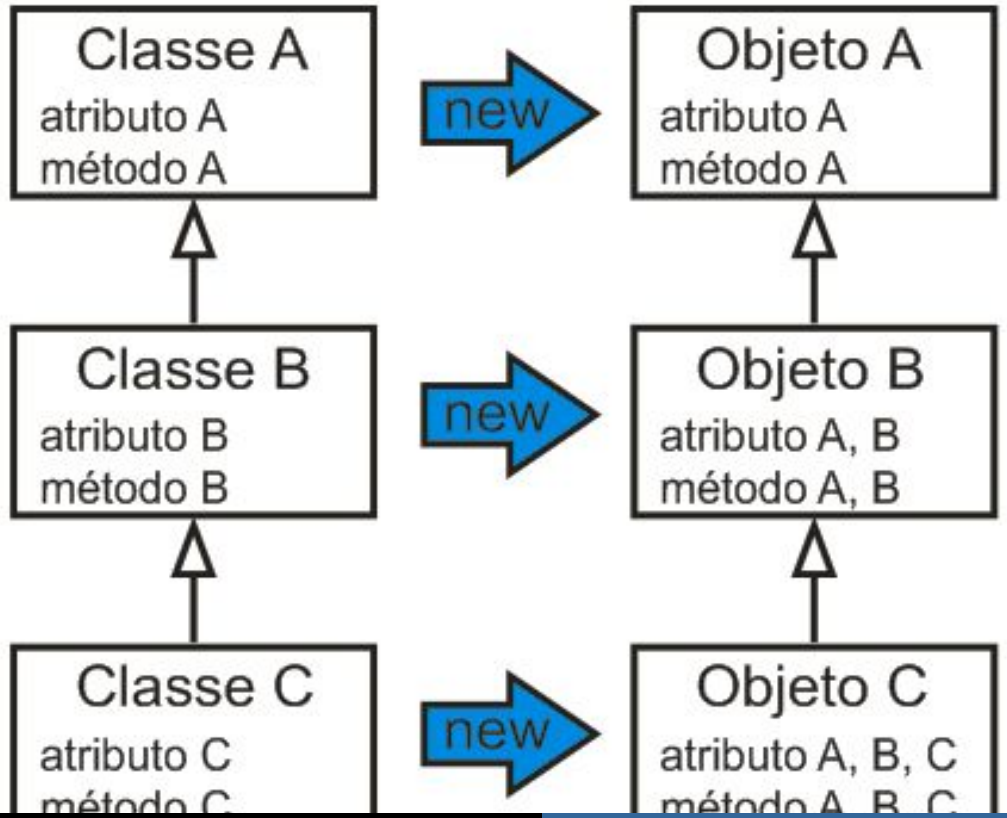
## Herança

```
1  public class Main {
2      public static void main(String[] args) {
3          Pessoa p = new Pessoa("João");
4          System.out.println(p.getNome());
5
6          Aluno a = new Aluno("Maria", "2008G55");
7          System.out.println(a.getNome() + "-" +
               a.getMatricula());
8
9          Professor prof = new Professor("José", 2500.5f);
10         System.out.println(prof.getNome() + "-" +
               prof.getSalario());
11     }
12 }
```

# Herança e polimorfismo

## Herança

- Um objeto possuirá os atributos e métodos pertencentes a classe de que é instância e todas as suas super-classes.



# Herança e polimorfismo

## Herança

```
1 public class AlunoEspecial extends Aluno {
2     public AlunoEspecial(String nome, String matricula) {
3         super(nome, matricula);
4     }
5 }
```

# Herança e polimorfismo

## Exercícios

**Exercício**

- Quais atributos e métodos podem ser utilizados em um objeto de AlunoEspecial?

# Herança e polimorfismo
## Herança

- Modificadores:
  - final (para classe) - não permite que uma classe possua subclasses
  - protected (para métodos e atributos) - permite acesso a classes em um mesmo pacote ou para as subclasses

# Herança e polimorfismo

## Herança

```
1 public class Animal {
2 }
```

```
1 public class Vertebrado extends Animal {
2 }
```

```
1 public class Mamifero extends Vertebrado {
2 }
```

```
1 public final class Humano extends Mamifero {
2 }
```

# Herança e polimorfismo

## Exercícios

### Exercício

- Verifique o que ocorre ao se tentar criar uma subclasse de Humano.

# Herança e polimorfismo

## Herança

```
1 package casa;
2 public class Pessoa {
3     protected String nome;
4 }
```

```
1 package casa;
2 public class Filho extends Pessoa {
3     public void exemplo() {
4         nome = "teste";
5     }
6 }
```

# Herança e polimorfismo

## Herança

```
1 package rua;
2 import casa.Pessoa;
3 public class FilhoNaRua extends Pessoa {
4     public void exemplo() {
5         nome = "teste2";
6     }
7 }
```

```
1 package rua;
2 import casa.Pessoa;
3 public class Desconhecido {
4     public void exemplo() {
5         Pessoa p = new Pessoa();
6         p.nome   = "Teste3";
7     }
8 }
```

# Herança e polimorfismo

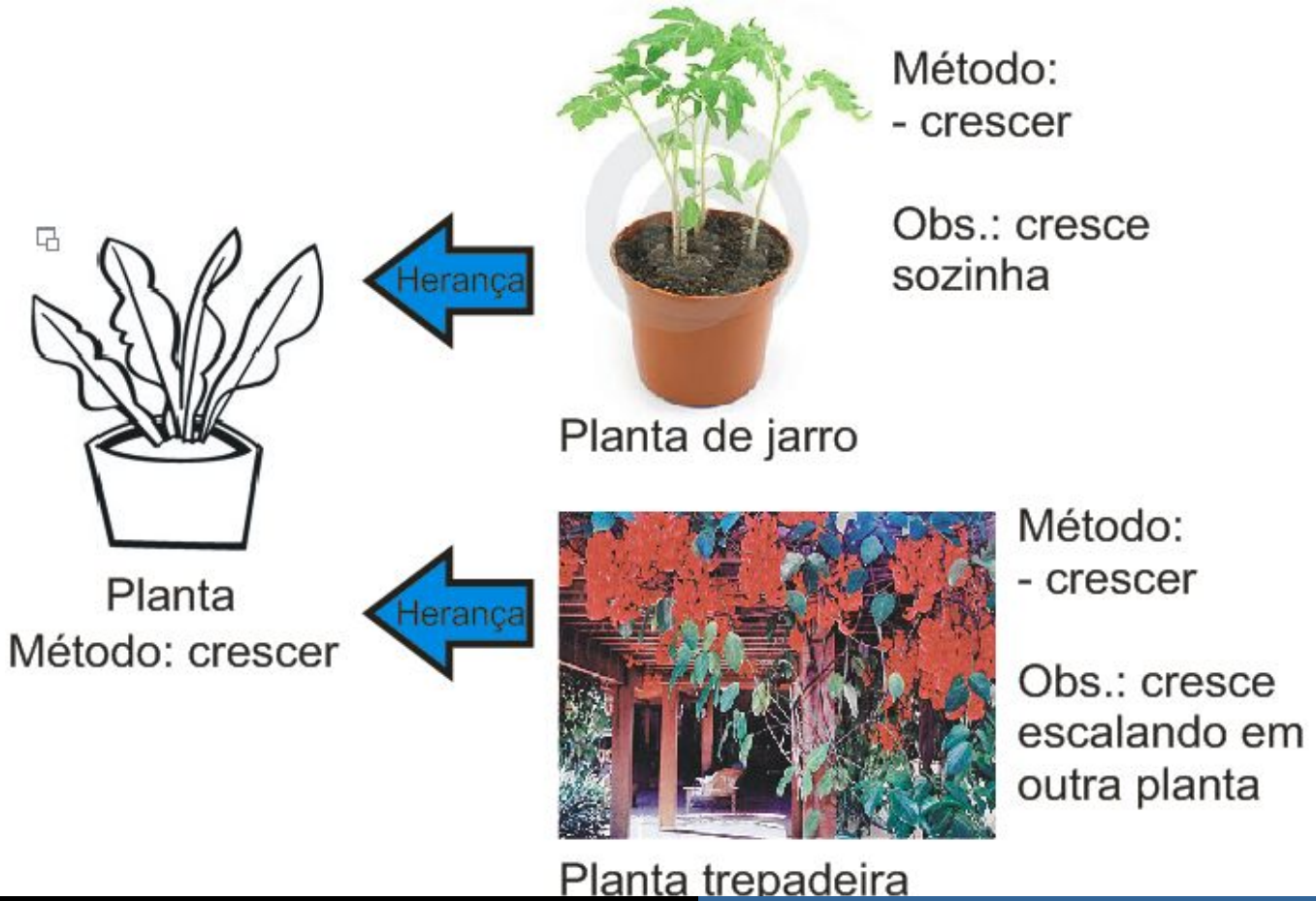
## Polimorfismo

- Poli = “muitos”; Morfo - “forma”.
- Em O.O. polimorfismo se refere a capacidade de objetos responderem de formas distintas a uma mesma requisição

# Herança e polimorfismo

## Polimorfismo

Uma analogia...



# Herança e polimorfismo

## Polimorfismo

- Polimorfismo está ligado ao conceito de sobreescrita (*overriding*) e sobrecarga (*overloading*).
- Sobreescrita ocorre quando uma subclasse reimplementa um método existente na superclasse.
- Sobrecarga ocorre quando um método com mesmo nome possui diferentes parâmetros (quantidade e/ou tipos).

# Herança e polimorfismo

## Polimorfismo - Sobreescrita

```
1 public class Pessoa {
2     // Todos os atributos e métodos anteriores permanecem
3
4     public String toString() {
5         return nome;
6     }
7 }
```

```
1 public class Aluno extends Pessoa {
2     // Todos os atributos e métodos anteriores permanecem
3
4     @Override
5     public String toString() {
6         return getNome() + "-" + matricula;
7     }
8 }
```

# Herança e polimorfismo

## Polimorfismo - Sobreescrita

```
1 public class Main {
2     public static void main(String[] args) {
3         Pessoa p = new Pessoa("João");
4         System.out.println(p.toString());
5
6         Aluno a = new Aluno("Maria", "2008G55");
7         System.out.println(a.toString());
8     }
9 }
```

# Herança e polimorfismo

## Polimorfismo - Sobrecarga

```
1  public class Aluno extends Pessoa {
2      private String matricula;
3
4      public Aluno(String nome) {
5          super(nome);
6      }
7
8      public Aluno(String nome, String matricula) {
9          super(nome);
10         this.matricula = matricula;
11     }
12
13     // Métodos...
14 }
```

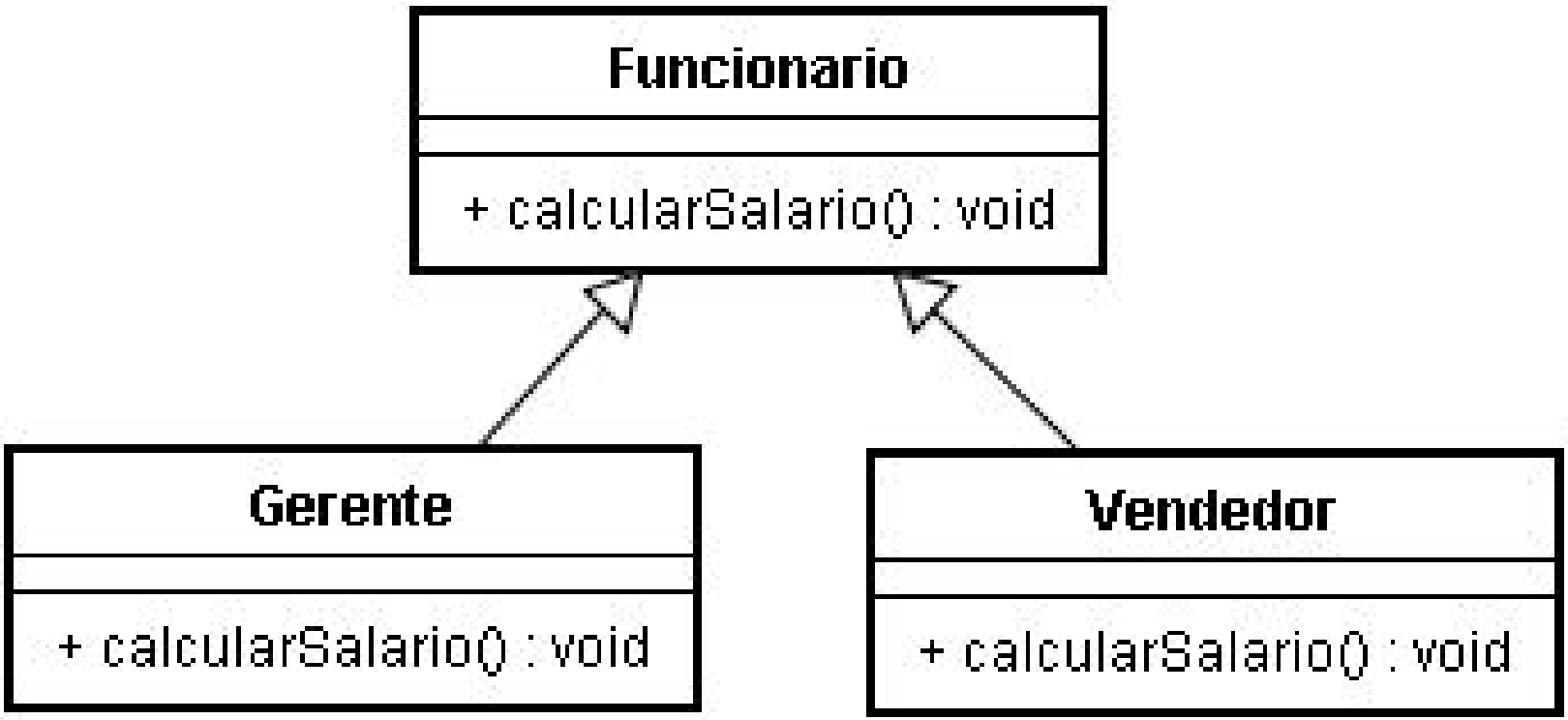
# Herança e polimorfismo

## Polimorfismo

- Uma empresa possui diversos funcionários, todos os funcionários recebem salários. Diferentes cargos possuem formas diferentes de calculo de pagamento.
  - Gerente tem seu salário baseado em um percentual sobre o aumento de vendas total.
  - Vendedores recebem de acordo com o número de vendas efetuado por ele.

# Herança e polimorfismo

## Polimorfismo



# Herança e polimorfismo
## Polimorfismo

- Modificadores:
  - final (para método) - não permite que o método seja sobreescrito

# Herança e polimorfismo
## Modificador abstract

Modificador: abstract

- Aplicável: Classes e métodos
- Descrição:
  - classes - não permite a criação de objetos e pode conter métodos abstratos
  - métodos - não podem ser implementados

# Herança e polimorfismo

## Modificador abstract

```
1 public abstract class Pessoa { // classe abstrata
2     private String nome;
3     public abstract String descricao(); // metodo abstrato
          (sem implementação)
4     public String getNome() { // metodo concreto (com
          implementação)
5         return nome;
6     }
7 }
```

```
1 public class Aluno { // classe concreta
2     // Obrigatória a implementação (ou postergar a
          implementação)
3     public String descricao() {
4         return "Aluno: " + getNome();
5     }
6 }
```

# Herança e polimorfismo
## Classes Abstratas

- As classes que foram implementadas até então são ditas "Classes concretas"
- Classes concretas e classes abstratas são diferentes em dois aspectos
- Classe concreta
  - Pode ser instanciada
  - Apenas métodos com implementação são permitidos (concretos)
- Classe abstrata
  - Não pode ser instanciada
  - Métodos com implementação (concretos) e métodos sem implementação (abstratos) são permitidos

# Herança e polimorfismo

## Classes Abstratas

```
1 public class ExemploConcreta {
2
3 }
```

```
1 public abstract class ExemploAbstrata {
2
3 }
```

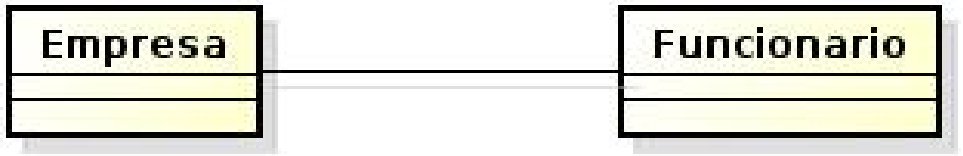
# Herança e polimorfismo

## Classes Abstratas

```
1 public class Main {
2     public static void main(String[] args) {
3         // Classe concreta pode ser instanciada
4         ExemploConcreta ec = new ExemploConcreta();
5
6         // Classe abstrata não pode ser instanciada
7         ExemploAbstrata ec = new ExemploAbstrata(); // ERRO
8     }
9 }
```

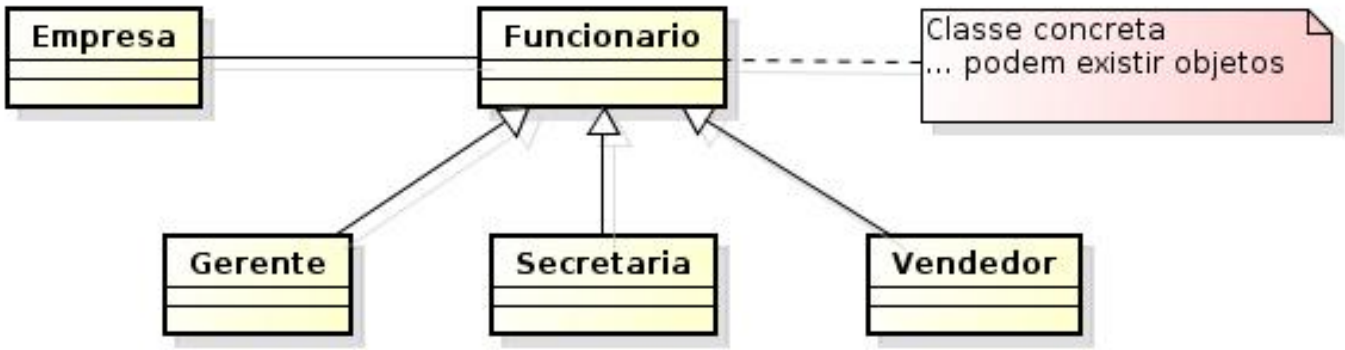
# Herança e polimorfismo
## Classes Abstratas

Uma empresa possui funcionários...



# Herança e polimorfismo
## Classes Abstratas

Uma empresa possui funcionários que podem ser funcionários em geral, gerentes, secretárias ou vendedores



# Herança e polimorfismo
## Classes Abstratas

```
1 public class Funcionario {
2 }
```

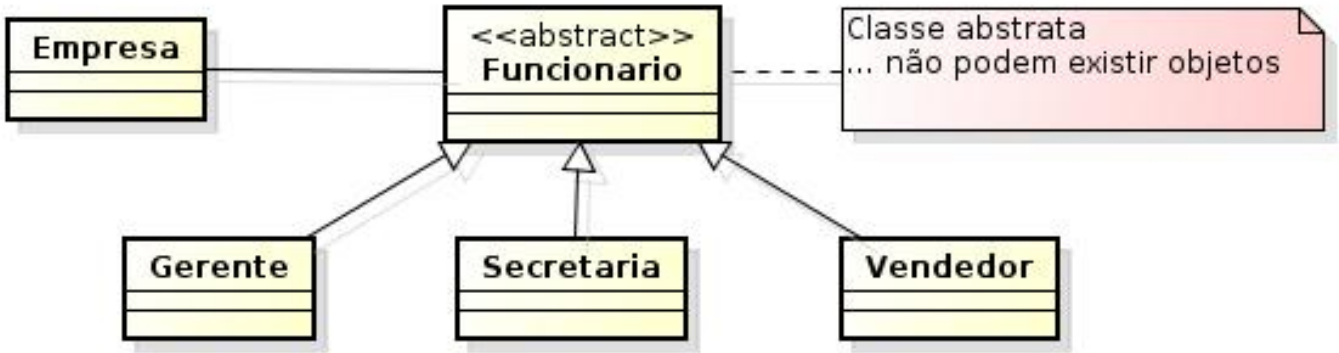
```
1 public class Gerente extends Funcionario {
2 }
```

```
1 public class Secretaria extends Funcionario {
2 }
```

```
1 public class Vendedor extends Funcionario {
2 }
```

# Herança e polimorfismo
## Classes Abstratas

Uma empresa possui funcionários que só podem ser gerentes, secretárias ou vendedores



# Herança e polimorfismo
## Classes Abstratas

```
1 public abstract class Funcionario {
2 }
```

```
1 public class Gerente extends Funcionario {
2 }
```

```
1 public class Secretaria extends Funcionario {
2 }
```

```
1 public class Vendedor extends Funcionario {
2 }
```

# Herança e polimorfismo
## Classes Abstratas

- Comportamento padrão pode ser implementados em uma classe abstrata
- Comportamento comum a diversas classes, mas que é executado de maneira diferenciada pode ser descrito em uma classe abstrata (em métodos abstratos)
- Classes concretas que herdam de uma abstrata devem implementar o comportamento
- Comportamento padrão é implementado (Métodos concretos)
- Comportamento comum é descrito (Método abstrato)

# Herança e polimorfismo

## Classes Abstratas

```
1  public abstract class ExemploAbstrata {
2
3      // Comportamento comum a várias classes,
4      // mas sem padrão de execução
5      public abstract void exemploAbstrato();
6
7      // Comportamento comum a várias classes,
8      // com padrão de execução
9      public void exemploConcreto () {
10         // implementação
11     }
12
13 }
```

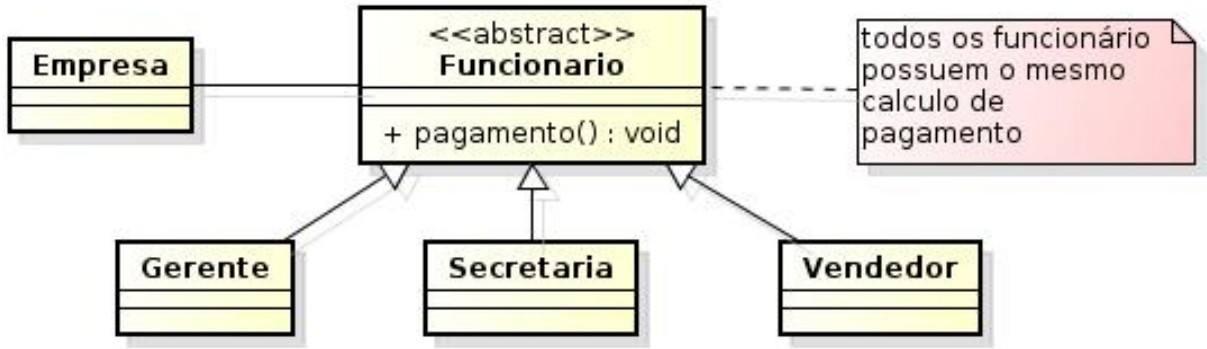
# Herança e polimorfismo

## Classes Abstratas

```
1 public class ExemploConcreta extends ExemploAbstrata {
2
3     // Deve implementar o comportamento comum
4     public abstract void exemploAbstrato() {
5         // implementação
6     }
7 }
```

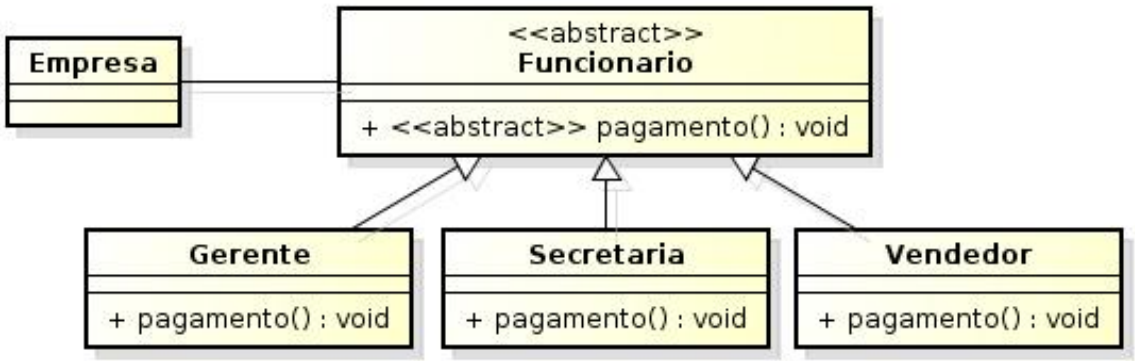
# Herança e polimorfismo
## Classes Abstratas

Todos os funcionários possuem o mesmo cálculo de pagamento



# Herança e polimorfismo
## Classes Abstratas

Cada tipo de funcionário possui uma forma de cálculo de pagamento

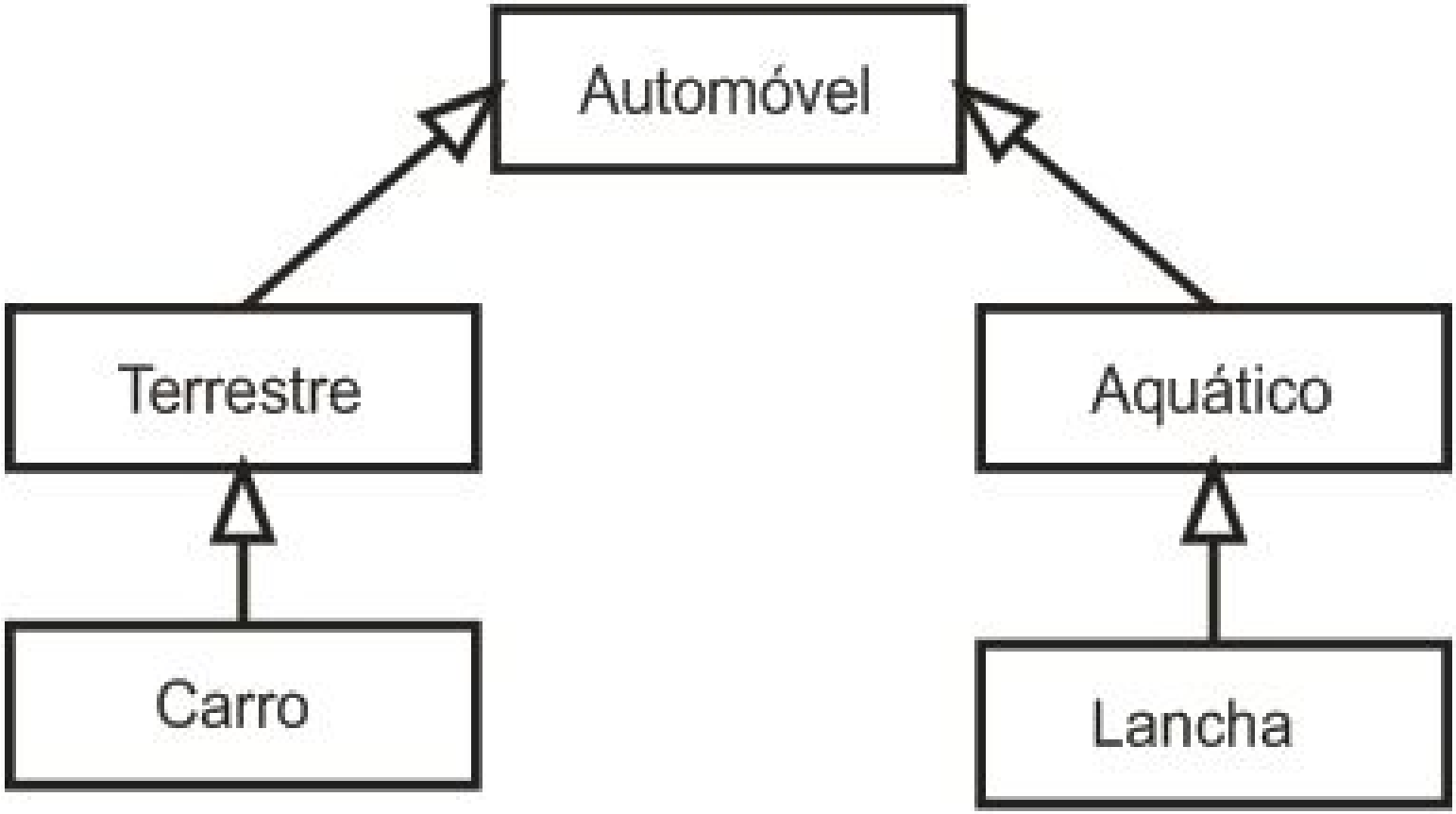


# Herança e polimorfismo
## Interface

- Existem basicamente dois tipos de herança:
  - simples - uma classe só possui uma única super-classe
  - múltipla - uma classe possui mais de uma super-classe

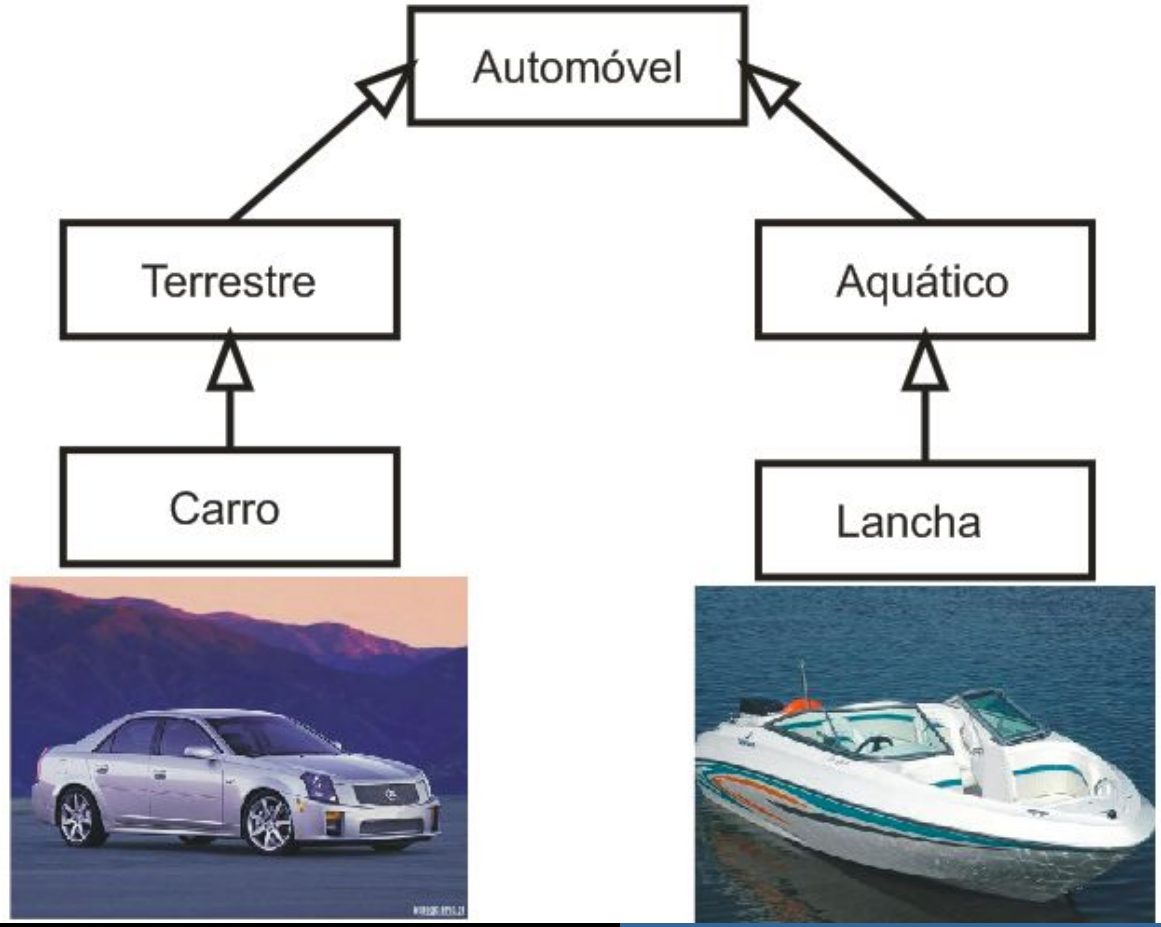
# Herança e polimorfismo
## Interface

Herança simples é suficiente...



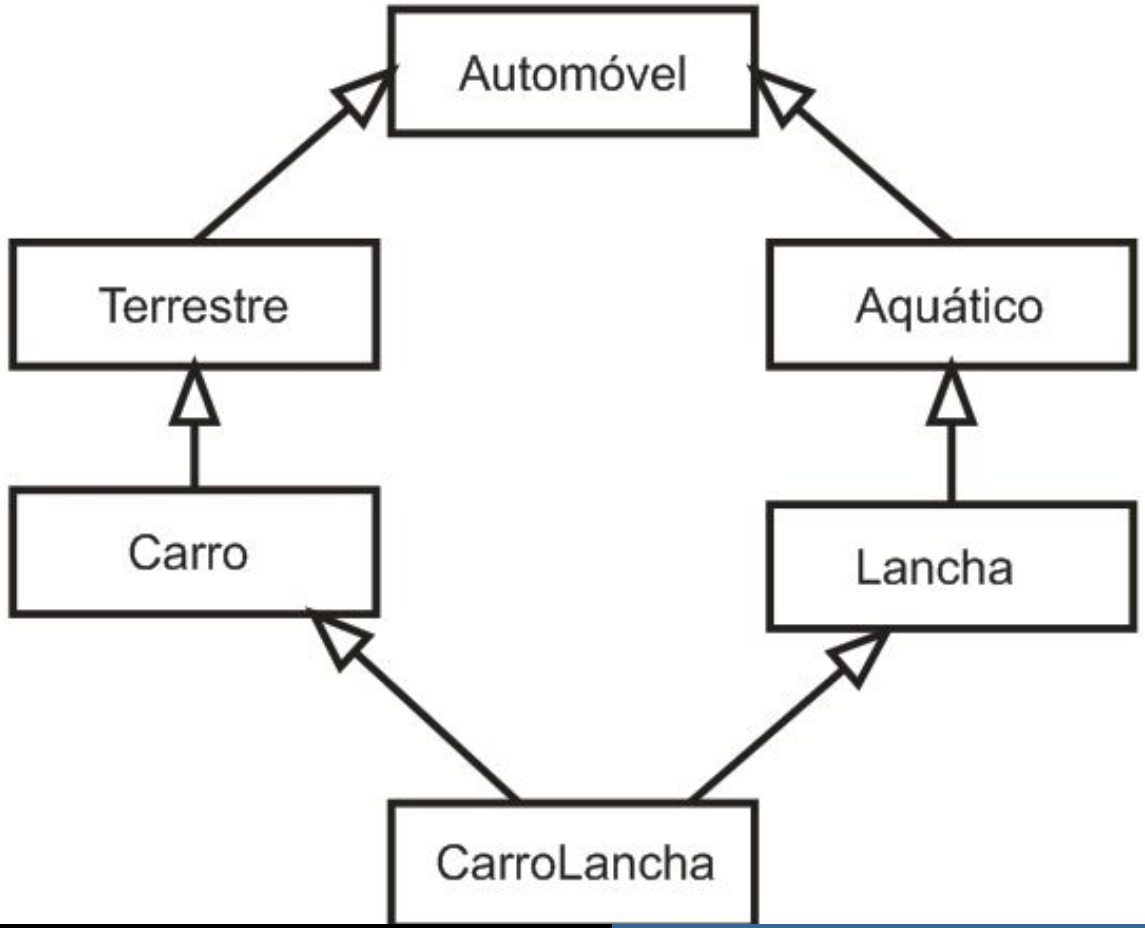
# Herança e polimorfismo
## Interface

Herança simples é suficiente...



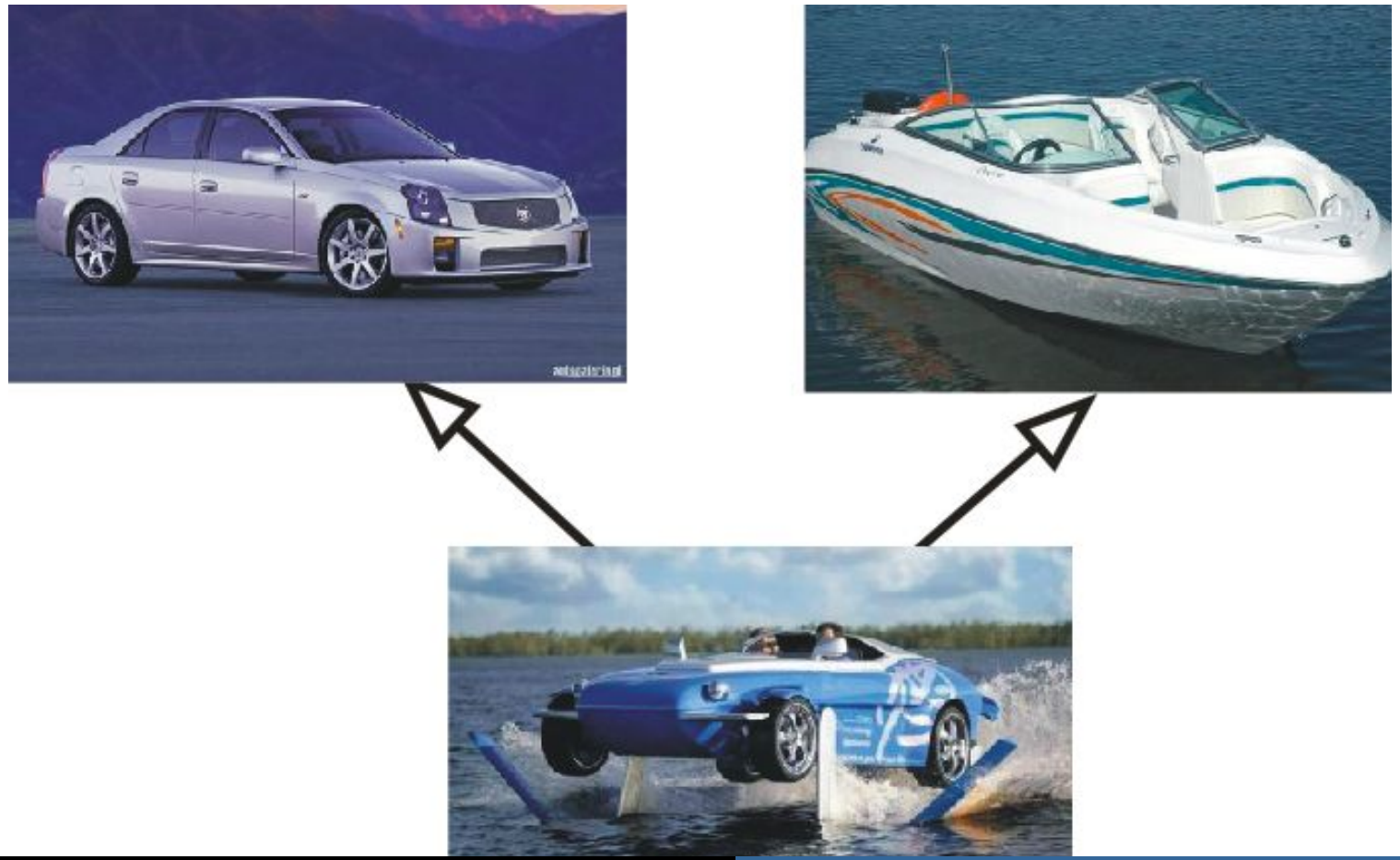
# Herança e polimorfismo
## Interface

Herança simples não é suficiente...



# Herança e polimorfismo
## Interface

Herança múltipla resolve o problema ...



# Herança e polimorfismo
## Interface

Problema com herança múltipla...

Atributos:
- Rodas
- Volante

Métodos:
- Acelerar
- Frear
- Ligar

Atributos:
- Rodas
- Hélice
- Volante (??)

Métodos:
- Acelerar (??)
- Frear (??)
- Ligar (??)

Atributos:
- Hélice
- Volante

Métodos:
- Acelerar
- Frear
- Ligar

# Herança e polimorfismo
## Interface

- Uma interface é equivalente a uma classe abstrata onde todos os métodos são abstratos
- Uma classe pode herdar de uma única classe
- Uma classe pode implementar várias interfaces

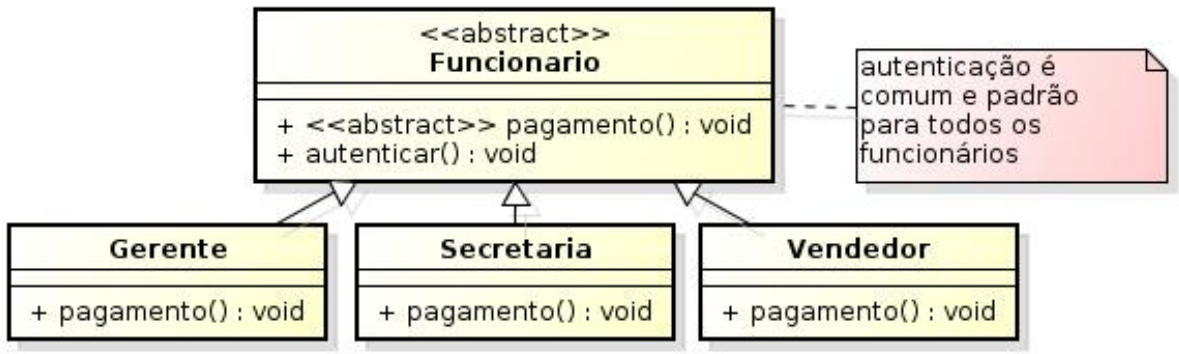
# Herança e polimorfismo
## Interface

```
1 public interface ExemploInterface {
2     // pertence a uma interface,
3     // então não possui implementação
4     public void exemploMetodo();
5
6 }
```

# Herança e polimorfismo
## Interface

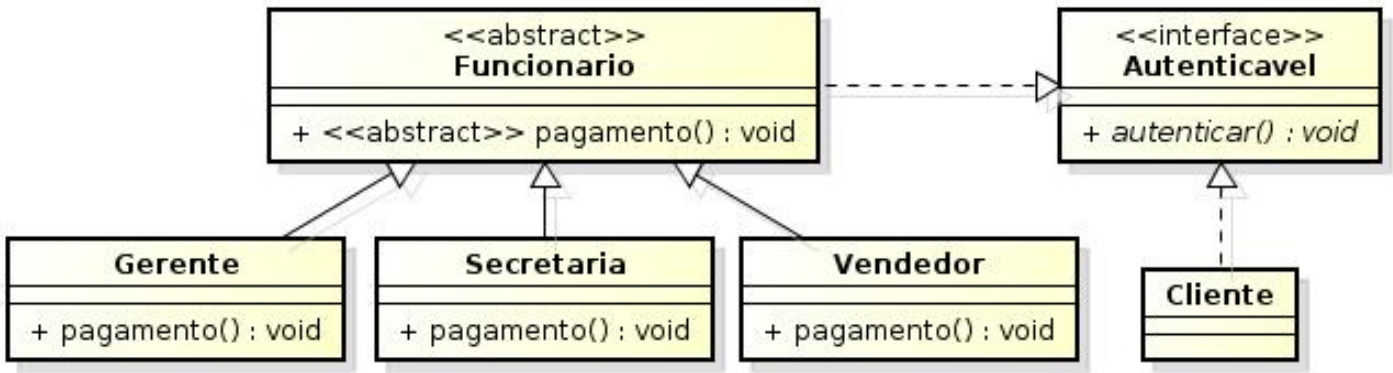
```
1 public class Exemplo implements ExemploInterface {
2
3     // Deve implementar o comportamento comum
4     public void exemploMetodo() {
5         // implementação
6     }
7
8 }
```

# Herança e polimorfismo
## Interface



# Herança e polimorfismo

## Interface



# Herança e polimorfismo

## Exercícios

### Exercício

1. Implemente as classes Conta, ContaPoupança e ContaCorrente, utilizando os conceitos de encapsulamento, herança e polimorfismo e as seguintes restrições:
   - toda conta possui um número de identificação e um saldo
   - para toda conta é possível realizar as operações de depósito, saque, e observar o valor do saldo
   - para toda conta nas operações de depósito, não deve ser possível depositar valores negativos
   - para toda conta nas operações de saque, não deve ser possível sacar valores negativos
   - para contas do tipo poupança, não deve ser possível sacar um valor caso este seja maior que o saldo existente na conta
   - para contas do tipo corrente, nas operações de saque que resultem em um saldo negativo, deve ser cobrada uma taxa de R$ 5,00

# Herança e polimorfismo

## Exercícios

**Exercício**

1. Implemente a hierarquia Conta - ContaCorrente - ContaPoupanca descrita nas aulas anteriores fazendo uso de classes e métodos abstratos

# Herança e polimorfismo

## Resumo

### Resumo

- Herança: Capacidade de uma classe possuir os dados e ações de uma outra classe mais geral
- Polimorfismo: Capacidade de tratar uma ação de maneiras diferentes
- Classes abstratas possibilitam reuso com a descrição de comportamentos concretos a abstratos
- Interfaces são similares a classes abstratas, e possibilitam "herança múltipla"

# Leituras recomendadas

Caelum

*Java e Orientação a Objetos*, 2011.

Capítulo 7: Orientação a Objetos - herança, reescrita e polimorfismo

D. J. Barnes e M. Kolling

Programação Orientada a Objetos com Java

Capítulo 8: Aperfeiçoando estruturas com o uso de herança

Capítulo 9: Mais sobre herança

Rafael Santos

*Introdução à programação orientada a objetos usando Java*, 2003.

Capítulo 8: Reutilização de classes

# Leituras recomendadas


Caelum

*Java e Orientação a Objetos*, 2011.
Capítulo 9: Orientação a Objetos - Classes Abstratas
Capítulo 10: Orientação a Objetos - Interfaces

D. J. Barnes e M. Kolling

Programação Orientada a Objetos com Java
Capítulo 10: Técnicas de abstração adicionais

Rafael Santos

*Introdução à programação orientada a objetos usando Java*, 2003.
Capítulo 9: Classes abstratas e interfaces

# Herança e polimorfismo

## Perguntas?

**Alexandre de Andrade Barbosa**
*alexandre.barbosa@arapiraca.ufal.br*